

METHOD AND APPARATUS FOR

RESUMING EXECUTION OF A FAILED COMPUTER PROGRAM

FIELD OF THE INVENTION

The present invention generally relates to recovery from errors in computer programs, and more particularly to using alternative code to recover execution of a failed computer program.

BACKGROUND

Certain types of software errors are fatal to program execution. For example, a reference to a memory address that is beyond the address domain of a program will likely result in a fatal error. Certain timing or other transient conditions may also trigger fatal errors. While certain errors may be within the control of the software developer, the developer may be unable to guard against certain other errors in developing the software.

Program errors that are transient or timing-related are problematic to both the program user and to the program developer. From the user's point of view, failure of the program not only interrupts the task at hand, but may also result in lost work product. Reporting such transient errors is also difficult since the conditions required to reproduce the problem are likely to be unknown.

From the developer's point of view, much time may be spent trying to find the root cause of a problem where the root cause is external to the program. Furthermore, tracing the root cause of the problem may be difficult and time-consuming if there is there is scant information available for reproducing the problem.

A method and apparatus that address the aforementioned problems, as well as other related problems, are therefore desirable.

1

2 SUMMARY OF THE INVENTION

3 In various embodiments, a computer implemented method
4 is provided for recovery from fatal program errors. A
5 program is compiled using two compilers to generate first
6 and second sets of object code. Checkpoints are
7 identified in the program, and checkpoint code is
8 generated for execution at the checkpoints. If execution
9 of the first set of object code fails, checkpoint data is
10 recovered and execution of the program is resumed using
11 either the first or second set of object code. In one
12 embodiment, the first set of object code is re-executed
13 before trying the second set of object code. In another
14 embodiment, the second set of object code automatically
15 executed upon failure of the first set of object code.

16 It will be appreciated that various other embodiments
17 are set forth in the Detailed Description and Claims which
18 follow.

20 BRIEF DESCRIPTION OF THE DRAWINGS

21 Various aspects and advantages of the invention will
22 become apparent upon review of the following detailed
23 description and upon reference to the drawings in which:

24 FIG. 1 is a flowchart of a an example process for
25 compiling a program in accordance with one embodiment of
26 the invention;

27 FIG. 2 is a block diagram that illustrates an example
28 data structure used in the management of checkpoints;

29 FIG. 3 is a block diagram that illustrates the code
30 that results from compilation of a program in accordance
31 with one embodiment of the invention; and

32 FIG. 4 is a flowchart of an example process for error
33 recovery.

34

1 DETAILED DESCRIPTION

2 In various embodiments, the present invention
3 provides a method and apparatus for generating alternative
4 code that supports recovery from a fatal program error.

5 FIG. 1 is a flowchart of a an example process for
6 compiling a program in accordance with one embodiment of
7 the invention. The process generally entails generating
8 two sets of object code segments using two different
9 compilers or code generators. The second set of object
10 code segments are available for execution in the event
11 that the a fatal program error is encountered in executing
12 the first set of segments. The segments of object code
13 are delineated by checkpoints that are identified by the
14 compiler.

15 At steps 202 and 204, the program source code is
16 compiled using two different compilers, and two sets of
17 object code that are functionally equivalent are created
18 from the compilation. For example, the compilers may be
19 different versions of the same compiler or compilers from
20 different vendors. For making the two sets of object code
21 interchangeable, checkpoints are identified and checkpoint
22 code is generated for the two sets of object code at step
23 206. Known computer platforms allow multiple compilers to
24 coexist and also allow inter-operation at the
25 function/procedure call level.

26 In one embodiment, the checkpoints are identified and
27 the code generated as described in the co-pending patent
28 application entitled, "Compiler-based Checkpointing for
29 Support of Error Recovery" by Ziegler et al. and filed on
30 October 31, 2000, which has patent/application number
31 ***** and attorney docket number 10001159, is commonly
32 assigned to the assignee of the present invention, and the
33 contents of which are incorporated herein by reference.
34 Each segment of code is delineated by a checkpoint, as
35 determined by the compiler. The checkpoints are points in

1 the code where the state of the program is stored so that
2 execution can be recovered at the point in the program
3 following the checkpoint. For example, convenient places
4 for checkpoints are procedure boundaries.

5 At step 208, the code that performs the checkpointing
6 is generated and combined with the intermediate code that
7 was generated from the user's source code. Along with the
8 checkpointing code, a data structure is created for
9 storage of the checkpoint data at step 208.

10

11 FIG. 2 is a block diagram that illustrates an example
12 data structure used in the management of checkpoints in
13 accordance with one embodiment of the invention. To save
14 storage space, two checkpoint data sets 260a and 260b are
15 maintained.

16 Checkpoint data are alternately stored in checkpoint
17 data sets 260a and 260b for consecutive checkpoints. For
18 example, at time t1, checkpoint data set 260a references
19 checkpoint 262 and checkpoint data set 260b references
20 checkpoint 264. At time t2 after program execution
21 completes checkpoint 266, checkpoint data set 260a
22 references checkpoint 266, and checkpoint data set 260b
23 references checkpoint 264.

24 Timestamps or commit flags may be used in alternative
25 embodiments to indicate which of the checkpoint data sets
26 is to be used in recovery. The timestamp scheme involves
27 writing a timestamp to a checkpoint data set when the
28 storage of state information in the checkpoint data set is
29 complete. Thus, the later of the two timestamps indicates
30 which of checkpoint data sets 260a or 260b is to be used
31 in recovery. The commit flag scheme involves a flag that
32 indicates which of checkpoint data sets 260a or 206b is to
33 be used in recovery.

34

1 FIG. 3 is a block diagram that illustrates the code
2 that results from compilation of a program in accordance
3 with one embodiment of the invention. One purpose for
4 compiling code in the manner taught herein is to enable
5 recovery from fatal program errors. Block 102 represents
6 program source code that is to be compiled and is
7 comprised of n segments of source code. Checkpoints are
8 used to delineate the multiple segments. A checkpoint is
9 a location in the code at which execution can recommence
10 should the program encounter a fatal error. At each
11 checkpoint, the state of data elements used by the program
12 are stored so that in the event of program failure the
13 state information can be recovered and execution resumed
14 immediately after the checkpoint from which the state was
15 recovered. In various embodiments the checkpoints can be
16 user-programmed or identified by the compiler using
17 recognized techniques.

18 Compilation of the program source code results in
19 program object code 104 that includes two sets of object
20 segments, object segments 1- n and object segments 1- n' .
21 The object segments in each set correspond to the source
22 segments of program source code 102.

23 Each set of object segments is code that is generated
24 in compiling the source code with different compilers or
25 code generators. In other words, object segments 1- n are
26 generated by a first compiler, and object segments 1'- n'
27 are generated by a different compiler. If the program
28 fails during execution of segment i , for example, then the
29 state of the checkpoint data can be recovered from
30 checkpoint that precedes segment i and execution can
31 resume at segment i' .

32
33 FIG. 4 is a flowchart of an example process for error
34 recovery in accordance with one embodiment of the present
35 invention. The process generally entails recovering from

1 a fatal program error by restoring checkpoint data and
2 resuming execution of the program. In resuming execution,
3 the code of the first set of segments is retried. If the
4 failure is repeated, execution of the program is resumed
5 using the alternative set of segments.

6 The process begins when a program error has been
7 detected by the operating system, for example. At step
8 304, checkpoint data is restored, and at step 306 the
9 program counter is reset to the selected checkpoint. The
10 segment of object code from the first set of segments is
11 re-executed at step 308. If the program executes the
12 segment without error, decision step 310 and step 312
13 illustrate that the program continues with execution of
14 the segments from the first set.

15 If an error recurs in executing the segment of code
16 from the first set, control is directed to decision step
17 314, which determines whether the alternative code should
18 be tried. In one embodiment, the first set of segments of
19 code may be re-executed a selected number of times before
20 trying execution of the alternative code. Control is
21 directed to step 316 when the decision is made to execute
22 the alternative code.

23 At step 316, checkpoint data is restored, and at step
24 318, the address of the segment of object code from the
25 second set is selected for execution. That is, the
26 program counter is loaded with a program address of a
27 segment in the second set. At step 320, the alternative
28 segment of object code is executed, and execution of the
29 segments of code from the second set continues at step
30 322.

31 Returning now to decision step 314, if the decision
32 is made to not execute the alternative code, control is
33 directed to decision step 324. Decision step 324 tests
34 whether execution of the segment from the first set of
35 segments should be attempted again. If so, control is

1 returned to step 304 to restore the check point data and
2 try again. Otherwise, the program is exited with an
3 error.

4 The present invention is believed to be applicable to
5 compilers for a variety of programming languages. Other
6 aspects and embodiments of the present invention will be
7 apparent to those skilled in the art from consideration of
8 the specification and practice of the invention disclosed
9 herein. It is intended that the specification and
10 illustrated embodiments be considered as examples only,
11 with a true scope and spirit of the invention being
12 indicated by the following claims.

13